APLIKASI PEMROGRAMAN DINAMIS UNTUK MEMAKSIMALKAN PELUANG MEMENANGKAN PERMAINAN "PIG"

Stevie Giovanni 13506054

Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika
Insitut Teknologi Bandung
Jl. Ganesha 10, Bandung
e-mail: steviegiovanni@gmail.com, if16054@students.if.itb.ac.id, steviegiovanni@yahoo.com, steviegiovanni@hotmail.com

ABSTRAK

Banyak algoritma atau strategi pemecahan masalah yang diterapkan pada permainan-permainan sederhana. Hal ini dikarenakan, permainan-permainan sederhana seperti empat-satu, blackjack, monopoli, dan semacamnya dianggap sangat cocok untuk merepresentasikan masalah-masalah yang dapat dipecahkan dengan algoritma-algortima tersebut. Algoritma dynamic programming atau pemrograman dinamis merupakan salah satu algoritma yang masalahnya dapat direpresentasikan dengan permainan sederhana. Masalah-masalah yang dapat dipecahkan menggunakan pemrograman dinamis umumnya berhubungan dengan pencarian solusi optimum. Pada makalah ini penulis akan membahas mengenai aplikasi pemrograman dinamis untuk memaksimalkan peluang memenangkan permainan "Pig".

Tujuan penulisan makalah ini adalah sebagai tugas mata kuliah IF2251 Strategi Algoritmik. Dalam memilih topik makalah, selain waktu yang mendesak, penulis mempertimbangkan agar penulisan makalah ini juga bermanfaat memperdalam pengetahuan penulis mengenai algoritma yang dibahas. Dengan pertimbangan tersebut, penulis memutuskan membahas pemrograman dinamis dalam makalah ini.

Dalam penulisan makalah ini, penulis banyak mendapatkan masukan dari berbagai sumber yang berkaitan. Sumber utama yang menjadi referensi penulis adalah Diktat Kuliah IF2251 Strategi Algoritmik yang disusun oleh Ir. Rinaldi Munir,M.T. Selain dari sumber tersebut penulis juga menggunakan situs-situs pengetahuan sebagai tambahan.

Kata kunci: pemrograman dinamis, pig.

1. PENDAHULUAN

Permainan sederhana seperti empat-satu, blackjack, monopoli, catur, dan semacamnya telah populer sejak puluhan tahun yang lalu. Permainan-permainan ini tidak hanya menarik, tetapi juga mudah dimainkan dan mempunyai unsur kecerdasan dalam memainkannya. Seorang pemain catur harus pandai mengatur strategi untuk memenangkan permainan, pemain monopoli harus pintar melakukan transaksi untuk menjadi pemain terkaya, pemain blackjack dan empat-satu harus tahu kapan harus menghentikan permainan untuk memenangkannya.

Kebanyakan orang memandang permainan sederhana hanya sekedar permainan belaka. Namun pada kenyataannya, di balik kesederhanaan permainan tersebut, terdapat algoritma-algoritma tertentu yang dapat diterapkan untuk memenangkan permainan-permainan tersebut. Walaupun algoritma-algoritma ini tidak menjamin kemenangan seorang permain, dengan menggunakan algoritma tertentu, seorang pemain dapat meningkatkan peluang untuk memenangkan permainan.

2. APLIKASI PEMROGRAMAN DINAMIS DALAM PERMAINAN PIG

2.1 Permainan Pig

Permainan pig adalah salah satu permainan sederhana yang melibatkan dua orang pemain dan menggunakan sebuah dadu. Permainan dimainkan secara bergilir. Pada tiap giliran, pemain akan melemparkan dadu. Hasil dari pelemparan menentukan kelanjutan permainan.

a. Jika hasil pelemparan adalah 1, pemain tersebut tidak mendapat nilai dan pemain lain akan mendapatkan giliranya.
b. Jika hasil pelemparan adalah angka selain 1, hasil tersebut akan ditambahkan pada nilai total yang didapatkan pemain pada gilirannya tersebut. Pemain kemudian dapat memilih untuk melanjutkan melempar dadu atau berhenti.

c. Jika pemain memutuskan untuk berhenti, nilai total yang didapatkan pemain pada gilirannya akan ditambahkan ke nilai total yang didapatkan pemain dari giliran-giliran sebelumnya.

Permainan akan terus berlanjut sampai ada pemain yang mencapai nilai 100. Pemain yang pertama kali mencapai nilai 100 akan memenangkan permainan.

Masalah yang seringkali dihadapi pemain adalah kapan kita sebaiknya berhenti melempar dadu? Knizia [3] dalam bukunya memperkenalkan strategi "hold at 20" (berhenti melempar dadu jika nilai lebih besar atau sama dengan 20). Strategi ini didasarkan pada pengetahuan bahwa perbandingan kemunculan angka 1 (kehilangan segalanya) dan angka selain satu (nilai bertambah) adalah 1 banding 5. Nilai rata-rata tiap pelemparan adalah 4. Jika seorang pemain mempertaruhkan 20 angka, maka perbandingannya adalah 4 banding 20 atau 1 banding 5. Dengan begitu permainan baru dapat dikatakan *fair*.

Walaupun strategi "hold at 20" adalah strategi yang baik, bukan berarti strategi ini dapat terus digunakan. Jika nilai yang kita peroleh dan nilai lawan berturut-turut adalah 78 dan 99, kita tidak mungkin menerapkan strategi ini dan berhenti setelah nilai kita 98. Hal ini karena peluang untuk menang jika kita terus melempar akan lebih besar dibanding peluang menang jika kita menyerahkan giliran kepada lawan.

2.2 Memaksimalkan Peluang Menang

Asumsikan $P_{i,j,k}$ sebagai peluang pemain untuk menang dengan skor i, skor lawan j, dan nilai pemain pada giliran tersebut adalah k. Pada kasus dimana $i + k \geq 100$, kita dapatkan $P_{i,j,k} = 1$ karena pemain cukup berhenti melempar dadu dan menang. Pada kebanyakan kasus di mana $0 \leq i,j < 100$, dan $k < 100–i$, peluang seorang pemain untuk menang adalah

$$P_{i,j,k} = \max(P_{i,j,k,roll}, P_{i,j,,k,hold}) \quad (1)$$

di mana $P_{i,j,k,roll}$ dan $P_{i,j,k,hold}$ adalah peluang menang jika pemain melempar lagi atau bertahan. Kedua peluang ini didefinisikan sebagai berikut.

$$P_{i,j,k,roll}=((1–P_{j,i,0})+P_{i,j,k+2}+P_{i,j,k+3}+P_{i,j,k+4}+P_{i,j,k+5}+P_{i,j,k+6})/6 \quad (2)$$

$$P_{i,j,k,hold} = 1 – P_{j,i+k,0} \quad (3)$$

Peluang menang setelah mendapat hasil pelemparan 1 atau setelah berhenti melempar adalah peluang bahwa pemain lain tidak akan menang pada permulaan giliran selanjutnya.

Perhatikan bahwa memecahkan masalah optimasi ini tidak semudah yang kita bayangkan karena terdapat ketergantungan yang siklik antara variabel yang satu dengan yang lain. Sebagai contoh, $P_{i,j,0}$ bergantung pada $P_{j,i,0}$ yang bergantung pada $P_{i,j,0}$. Dengan kata lain, kita tidak dapat dengan mudah mengevaluasi peluang tahap demi tahap secara bottom-up seperti pada pemrograman dinamis.

2.3 Pemrograman Dinamis [1]

Pemrograman dinamis atau dynamic programming adalah metode pemecahan masalah di mana solusi diuraikan menjadi sekumpulan langkah atau tahapan sedemikian sehingga solusi dari persoalan dipandang dari serangkaian keputusan yang saling berkaitan.

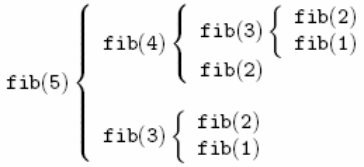
Penyelesaian masalah menggunakan pemrograman dinamis memiliki kemiripan dengan penyelesaian dengan algoritma greedy. Kesamaannya adalah kedua strategi penyelesaian masalah tersebut terdiri dari beberapa tahapan yang membentuk solusi dari permasalahan. Namun, terdapat perbedaan mendasar dari kedua strategi penyelesaian masalah ini. Pada algoritma greedy pengambilan keputusan setiap tahap hanya didasarkan pada informasi lokal di mana dynamic programming mengambil keputusan berdasarkan tahap an yang sudah dilalui sebelumnya. Hal ini menyebabkan algoritma greedy gagal memberikan solusi optimal untuk beberapa kasus tertentu.

Salah satu strategi terbaik untuk menemukan solusi optimal adalah dengan mengenumerasi semua solusi yang mungkin dibuat (exhaustive search). Namun cara ini tentunya akan menjadi tidak efektif jika terdapat benyak sekali kemungkinan. Di sinilah letak keunggulan pemrograman dinamis. Pemrograman dinamis mengandalkan memori untuk mengurangi jumlah komputasi. Hal ini akan ditunjukkan dengan contoh persoalan sederhana menghitung bilangan Fibonacci di bawah ini.

Sebagai ilustrasi sederhana, rumus untuk mendapatkan bilangan fibonacci ke-n adalah sebagai berikut :

$$\mathrm{fib}_n = \begin{cases} 1 & n = 1, 2 \\ \mathrm{fib}_{n-1} + \mathrm{fib}_{n-2} & n > 2 \end{cases} \quad (4)$$

Dengan pendekatan rekursif seperti ini, semakin besar nilai n, maka waktu yang diperlukan proses perhitungan akan meningkat secara eksponensial. Perhatikan proses perhitungan fib(5) berikut.



Gambar 1 Perhitungan fib(5)

Terlihat pada gambar bahwa untuk menghitung fib(5), dilakukan dua kali perhitungan fib(3). Hal ini menjadi lebih parah seiring bertambahnya nilai n.

Tabel 1 Pemanggilan rekursif fib(n)

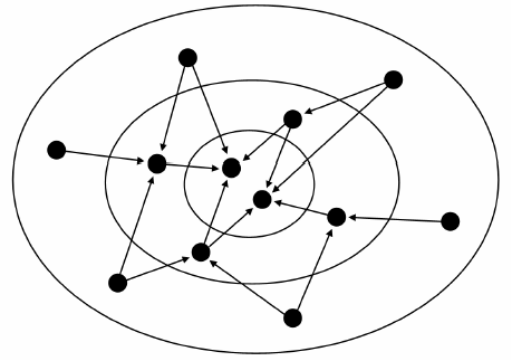
	Recursive Calls for `fib(n)`				
`n`	`fib(1)`	`fib(2)`	`fib(3)`	`fib(4)`	`fib(5)`
1	1	0	0	0	0
2	0	1	0	0	0
3	1	1	1	0	0
4	1	2	1	1	0
5	2	3	2	1	1
6	3	5	3	2	1
7	5	8	5	3	2
8	8	13	8	5	3
9	13	21	13	8	5
10	21	34	21	13	8
			...		
20	2584	4181	2584	1597	987
			...		
30	317811	514229	317811	196418	121393
			...		
40	39088169	63245986	39088169	24157817	14930352

Kita dapat menghindari hal ini dengan menyimpan nilai dari fib(n) yang sudah dihitung dan menggunakannya jika dibutuhkan. Inilah yang dilakukan dalam dynamic programming. Dynamic programming mengorbankan memori untuk mengurangi waktu proses.

Terdapat dua jenis pemecahan masalah dengan pemrograman dinamis, bottom-up atau top-down. Pada bottom-up, kita membentuk solusi tahap demi tahap dari tahap akhir ke tahap awal, sedangkan pada top-down kita melakukan hal sebaliknya.

2.4 Progressive Pig

Kita tidak dapat menyelesaikan pig dengan pemrograman dinamis karena ketergantungan antara state-state penyelesaian masalah yang dibentuk oleh pemrograman dinamis bersifat asiklik. Pemrograman dinamis membantu kita membentuk serangkaian solusi dari basis solusi ke solusi akhir yang didasarkan pada basis tersebut. Pada gambar berikut diilustrasikan pembentukan solusi pada dynamic programming. Setiap state pembentukan solusi bergantung hanya pada tahapan sebelumnya. Kebergantungan siklik pada permainan pig membuat kita tidak dapat membagi proses menjadi tahapan-tahapan tersebut.



Gambar 2 Membagi proses dynamic programming menjadi beberapa tahapan

Namun, dengan sedikit perubahan pada aturan permainan, kita dapat membuat kebergantungan antara variable bersifat asiklik. Caranya adalah dengan membuat tahapan tidak pernah berulang. Permainan pig yang telah dimodifikasi dinamakan progressive pig.

Progressive pig sangat mirip dengan pig tetapi terdapat pengecualian bahwa seorang pemain selalu mendapatkan nilai paling tidak 1 pada setiap giliran :

a. Jika hasil pelemparan adalah 1, pemain tersebut mendapat nilai 1 dan pemain lain akan mendapatkan giliranya.
b. Jika hasil pelemparan adalah angka selain 1, hasil tersebut akan ditambahkan pada nilai total yang didapatkan pemain pada gilirannya tersebut. Pemain kemudian dapat memilih untuk melanjutkan melempar dadu atau berhenti.
c. Jika pemain memutuskan untuk berhenti, nilai total yang didapatkan pemain pada gilirannya akan ditambahkan ke nilai total yang didapatkan pemain dari giliran-giliran sebelumnya.

Dengan perubahan aturan tersebut maka persamaan $P'_{i,j,k} = \max(P'_{i,j,k,roll} , P'_{i,j,k,hold})$, adalah

$$P'_{i,j,k,roll} = ((1 - P'_{j,i+1,0}) + P'_{i,j,k+2} + P'_{i,j,k+3} + P'_{i,j,k+4} + P'_{i,j,k+5} + P'_{i,j,k+6}) / 6 \quad (5)$$

$$P'_{i,j,k,hold} = 1 - P'_{j,i + \max(k,1),0} \quad (6)$$

2.5 Menyelesaikan Progressive Pig dengan Dynamic Programming

Untuk menyelesaikan progressive pig menggunakan dynamic programming, kita menyimpan score sementara menggunakan array 3D p[i][j][k] yang merepresentasikan nilai $P'_{i,j,k}$ hasil perhitungan dan roll[i][j][k] yang merepresentasikan tindakan optimal untuk dilakukan pada state (i,j,k)

Berikut diberikan algoritma untuk memecahkan masalah progressive pig.

```
Deklarasi
 int goal;
 boolean[0..goal,0..goal,0..goal] computed
 double[0..goal,0..goal,0..goal] p
 boolean[0..goal,0..goal,0..goal] roll
```

Ketika menggunakan algoritma ini, kita menggunakan nilai goal sebagai parameter masukan. Setelah semua variable dideklarasi, kita menghitung semua peluang kemenangan untuk semua state.

```
for (int i = 0; i < goal; i++)
```

```
   for (int j = 0; j < goal; j++)
      for (int k = 0; i + k < goal; k++)
         pWin(i, j, k)
      endfor
   endfor
endfor
```

Pada method pWin, kita memeriksa apakah seorang pemain telah menang, dan mengembalikan nilai peluang kemenangan 0 atau 1 tergantung pemain mana yang telah mencapai nilai 100. Selanjutnya kita memeriksa apakah peluang ini sudah pernah dihitung dan disimpan. Jika ya, maka kita cukup menggunakannya sebagai nilai kembalian. Inilah langkah dynamic programming kita. Langkah ini memastikan kita tidak melakukan komputasi yang tidak berguna.

```
function pWin(int i, int j, int k) : double
begin
   if (i + k >= goal) then
      return 1.0
   else if (j >= goal) then
      return 0.0
   else if (computed[i][j][k]) then
      return p[i][j][k]
   else
      <<hitung p[i][j][k] secara rekursif>>
      return p[i][j][k];
   endif
end
```

Untuk <<hitung p[i][j][k] secara rekursif>> kita cukup melakukan persamaan (5) dan (6)

```
{menghitung kemenangan jika terus melempar}
   double pRoll = 1.0 - pWin(j, i + 1, 0)
   for (int roll = 2; roll <= 6; roll++)
      pRoll += pWin(i, j, k + roll);
   endfor
   pRoll /= 6.0;

{menghitung peluang kemenangan jika
berhenti melempar}
   double pHold;
   if (k == 0) then
      pHold = 1.0 - pWin(j, i + 1, 0);
   else
      pHold = 1.0 - pWin(j, i + k, 0);
   endif

{melakukan aksi sesuai peluang maksimum}
   roll[i][j][k] = pRoll > pHold;
   if (roll[i][j][k]) then
      p[i][j][k] = pRoll;
   else
      p[i][j][k] = pHold;
   endif
   computed[i][j][k] = true;
```

Akhirnya, kita tambahkan algoritma untuk menampilkan hasil. Pertama kita mencetak peluang pemain pertama untuk menang dengan permainan optimal. Lalu untuk setiap pasangan i,j, kita menampilkan nilai k di mana pemain tersebut harus melakukan tindakan (terus melempar atau berhenti)

```
write("p[0][0][0] = ", p[0][0][0])
write("i\tj\t Aksi berubah ketika k =")
for (int i = 0; i < goal; i++)
   for (int j = 0; j < goal; j++)
      int k = 0;
      write(i,"\t",j,"\t",<<aksi (i,j,k)>>)
      for (k = 1; i + k < goal; k++)
         if(roll[i][j][k]!=roll[i][j][k-1])
            write(k," ",<<aksi (i,j,k)>>)
         endif
      endfor
   endfor
endfor
```

Di mana <<aksi (i,j,k)>> adalah "roll" jika rol[i][j][k] = true dan "hold" jika false.

IV. KESIMPULAN

Dari penulisan makalah ini penulis menyimpulkan beberapa hal :

1. Dynamic programming sangat efektif untuk memotong waktu proses dengan menangani kelebihan komputasi yang dilakukan algoritma-algoritma rekursif.
2. Dynamic programming mengorbankan kapasitas memori untuk memotong waktu proses. Memory ini digunakan untuk menyimpan hasil komputasi yang sudah pernah dilakukan.
3. Persoalan dynamic programming adalah persoalan yang dapat dibagi menjadi tahapan-tahapan proses di mana setiap tahapan bergantung pada tahapan sebelumnya.
4. Ketergantungan antar tahapan pada proses dynamic programming tidak boleh bersifat siklik.

REFERENSI

[1] Munir, Rinaldi.2005. *"Strategi Algoritmik"*. Departemen Teknik Informatika, Institut Teknologi Bandung.

[2] Todd W. Neller, Ingrid Russell, Zdravko Markov, "Solving the Dice Game Pig: an introduction to dynamic programming and value iteration",2005, hal 1-9.
[3] Reiner Knizia. Dice Games Properly Explained. Elliot Right-Way Books, Brighton Road, Lower Kingswood, Tadworth, Surrey, KT20 6TD U.K., 1999.
[4] http://cs.gettysburg.edu/projects/pig/index.html. Tanggal akses 19 Mei 2008 pukul 13.00.